Electronic Version

Stylesheet Version v1.1.1

Description

Flexible LUN/LBA Interface for Content Addressable Reference Storage

BACKGROUND OF INVENTION

FIELD OF INVENTION

[0001] The present invention relates generally to the field of content addressed storage. More specifically, the present invention is related to a logical block addressing (LBA) and logical unit number (LUN) interface for the manipulation of reference data.

DISCUSSION OF PRIOR ART

[0002] The benefits of content addressed storage (CAS) have long since been known and have been used to store, access, and protect reference data. CAS has been devised to manipulate this class of data known as reference data. Reference data refers to data that is written once and infrequently retrieved. Examples of data that fall in this class include x-rays, photos and videos, land titles, and finan-

cial documents. Reference data differs from transactional data in that reference data is not updated or retrieved with any sort of regular frequency. CAS relies on a scheme that hashes input reference data content to generate a unique object identifier.

[0003] With CAS, the write-once property is easily enforced if an object is modified, then it is stored as a new object because input reference data content will hash to another unique object identifier. Duplicates are also easily detectable since a user inputting reference data content identical to content already stored on the system will receive an identical object identifier. If an identical object identifier is received, then a duplicate object will not be created. CAS also allows the verification of the content of stored objects during retrieval. When an object is retrieved, its content is rehashed and checked against a previous hash, generated when the object was initially stored. CAS also has a larger address space than an LUN/LBA interface.

[0004] Logical block addressing (LBA) is a method used to translate cylinder, head, and sector information of a disk drive into addresses that can be used by an operating system during boot-up. Logical unit number (LUN) refers to a

unique number given to each peripheral device connected to a common bus. Commands sent to a particular type of bus (e.g., SCSI) distinguish devices on the basis of their LUNs.

[0005] The CAS interface was initially proposed by Quinlan et al in "Venti: A new Approach to Archival Storage" paper. Although CAS provides many benefits, a CAS interface also requires overlaying applications to be re-written using CAS API or using an OID interface, both of which mandate a change in the implementation of existing applications. This is in opposition to the more traditionally used LUN/LBA interface.

[0006] US 2003/0001900 A1 discloses a method for executing one or more operations in a computer for interfacing an associated user with a knowledge portal that is operatively associated with a plurality of data objects in a data store. Cabanes et al. also discloses a user interface method that includes the steps of receiving user input; updating, based on received user input, one of a current object identity, a preview object identity, and a knowledge map parameter which is updated based on a current object identity. Storing user input requires placement of a document into context with respect to other information stored and

available on an electronic information system. This method is useful in the context of searching and content classification. There is limited or no provision in this reference for content addressing or hashing and write properties including write-once, write-many, and write-many with versioning.

[0007] US 6292880 discloses a method for caching information objects where information objects are stored according to the hash of an object's filename and its content. The disclosed method provides for an object store that supports multiple different versions of targeted alternates for the same name and storage of objects without content duplication. However, explicit mention of a method to write object content to a storage area, and re-write different object content in that same storage area is not made. Mattis et al. requires additional mappings from name keys to vectors of alternates as well as from content keys to object content data.

[0008] US 4064494 discloses a content addressable memory that can non-destructively be read and which is adapted to respectively store true and inverse information data. While this reference provides electronic architectures supporting information storage and comparison, there is no premise

for interfacing overlying applications with the electronic hardware existing on a lower level. In addition, no mention is made of a unique identifier for each piece of information data. There is also no premise for upholding a write-many with versioning property.

[0009] JP 2000285013 discloses an interface between a CPU and an SDRAM capable of quickly reading out data from the SDRAM by control similar to that of a DRAM. The disclosed device, however, fails to provide for a means of content addressing or hashing and write properties including write-once, write-many, and write-many with versioning in the English translation of the abstract.

[0010] "A Distributed Repository for Immutable Persistent Objects" by Douglas Wiebe discloses a system for manipulating system model objects that provides for transactional, write-once storage. System model objects describe the structure and versions of software and are located using an expanding ring multicast search algorithm. This reference provides for a system and software that addresses stored data by structure rather than solely content.

[0011] Whatever the precise merits, features and advantages of the above cited references, none of them achieve or fulfills the purposes of the present invention. There exists a need

to provide the benefits of CAS with a write-once, a write-many, and a write-many property in a single system and method. It is also advantageous that a system and method provide a LUN/LBA to interface with existing content addressable memory systems.

SUMMARY OF INVENTION

[0012] The present invention is a system and method for content addressable storage with an LUN/LBA interface. Reference data is manipulated in a manner similar to a CAS system, where an object ID (OID) table is used to maintain OIDs generated for hashed LBAs.

[0013] The system of the present invention is comprised of three tiers. A first tier logic block is used to provide an LUN/LBA storage interface to application programs and to facilitate the writing and retrieval of reference data. A second tier logic block hashes reference data content to be written and stores a generated OID in a high-level OID table. A third tier logic block facilitates the writing of reference data to an LUN and the verification of the accuracy of reference data retrieved from a LUN. A second low-level OID table facilitates this process.

[0014] Access to an LUN/LBA storage area may be governed by the following properties; a write-once property wherein

the modification of an object results in the storage of a new object because content hashes will no longer match; a write-many property wherein multiple updates on a particular LUN/LBA are allowed; and lastly, a write-many with versioning property may be allowed wherein separate versions of a single object are chained together in an OID table.

[0015] Verification of the content of reference data occurs at the time of their retrieval. When an object is retrieved, its content is re-hashed and checked against a previously generated hash when the object was initially stored.

[0016] An LUN/LBA interface is utilized to obtain the benefits of the CAS interface without the necessity of re-writing an application interacting with an LUN/LBA so that it can communicate with a CAS API.

BRIEF DESCRIPTION OF DRAWINGS

[0017] Figure 1 illustrates a multi-tiered infrastructure of the present invention.

[0018] Figure 2 illustrates a high-level and a low-level OID table.

[0019] Figure 3 is a process flow diagram of first tier processing.

[0020] Figure 4 is a process flow diagram of second tier processing.

[0021] Figure 5 is a process flow diagram of third tier processing.

[0022] Figure 6 is a data flow diagram for writing content data.

[0023] Figure 7 is a data flow diagram for retrieving content data.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0024] While this invention is illustrated and described in a preferred embodiment, the device may be produced in many different configurations, forms and materials. There is depicted in the drawings, and will herein be described in detail, a preferred embodiment of the invention, with the understanding that the present disclosure is to be considered as an exemplification of the principles of the invention and the associated functional specifications for its construction and is not intended to limit the invention to the embodiment illustrated. Those skilled in the art will envision many other possible variations within the scope of the present invention.

[0025] An LUN/LBA interface is utilized to obtain the benefits of a content addressed storage (CAS) interface. Reference data is manipulated in a manner similar to a CAS system, where an object ID (OID) table is used to maintain OIDs generated for hashed LBAs.

[0026] A first tier logic block is used to provide an LUN/LBA stor-

age interface to application programs and to facilitate the writing of reference data. A second tier logic block hashes reference data to be written and stores an OID generated in a high-level table and a low-level table. A first table provides a mapping between a high-level LUN/LBA combination and an associated OID or OID list. A second table provides a mapping between a low-level LUN/LBA combination and an associated OID or OID list and also provides a counter for each OID. A third tier logic block facilitates writing content reference data to an LBA and verifies of the accuracy of reference data retrieved from an LUN. Reference data is written to a storage area in accordance with a write-once, write-many, or write-many with versioning property.

[0027] Referring now to Fig. 1, three-tiered infrastructure *100* is shown. Three-tiered infrastructure *100* comprises a first tier for LUN/LBA processing *104*, a second tier for OID processing *106*, a third tier for storage subsystem LUN/LBA processing *108*, a high-level OID table *110*, and a low-level OID table *112*.

[0028] In Fig. 2, two exemplary data structures of the present invention are shown. High-level OID table *200* contains two columns, a first column for an application-level LUN/LBA

combination *202* and a second column for an associated OID or OIDs *204*, the number of OIDs being dependant on a chosen access property. Low-level OID table *206* contains three columns, a first column for physical LUN/LBA value *208*, a second column for an associated OID or OIDs *210*, and a third column for a counter *212* that is incremented every time a different application program writes data that hashes to a particular OID value. Counter *212* ensures that an OID can only be deleted when all applications have stopped writing to a particular object. Note that physical LUN/LBA combination *208* differs fromapplication-level LUN/LBA combination *202* as it is an actual location of where reference data is stored in a storage subsystem, whereas application-level LUN/LBA combination *202* is a virtual location determined by application program data received at an interface. An application-level LUN/LBA combination *202* is location independent it allows an application or users of an application to group data. Mapping between an application-level LUN/LBA combination *202* and OID is set after content data has been hashed and an OID has been generated. Then, in high-level OID table *200*, for a particular LUN/LBA combination *202*, a generated OID value is stored. In a low-level

OID table *206*, an OID is written first. Once content data is written to a physical LUN/LBA combination, the value of a physical-level LUN/LBA combination to which content data was written is stored in low-level OID table *206*. Thus, mapping between a physical LUN/LBA combination to which content data was written and an OID corresponding to a hash of the written content data is established.

[0029] Referring now to Fig. 3, a first tier logic block is shown. First tier processing *300* involves interaction with application programs and provides an LUN/LBA storage interface. In first tier LUN/LBA processing block *300*, an application program data *302* provides reference data to be hashed, and for which an OID is generated. A comparison is made between a reference data OID value and OIDs stored in a high-level OID table *304*. From the comparison, a determination is made as to whether reference data for which an OID has been generated, has been previously written. The present invention allows for three different types of LUN/LBA access. The access type may be configured by a user (e.g. system administrator) and is determined subsequent to a previously mentioned determination step.

[0030] If a write-once property is enabled *306*, high-level OID table *200* is consulted to determine whether a particular

LUN/LBA *202* is associated with an OID *204*. If LUN/LBA *202* has been written to in the past, then LUN/LBA*202* will have a corresponding OID *204* in an OID column; otherwise, it will contain a default null value. If LUN/LBA has a null OID *308*, write operation is to LUN/LBA *332* is allowed. Subsequent to write operation to LUN/LBA *332*, an OID is written to high-level table *334*. If LUN/LBA has an OID *310*, then application program data *302* is denied a write operation to LUN/LBA *312*. A write-once property is enforced as a result of recognizing that if an object is modified, then it is stored as a new object because its content hash will be different. If a user has stored an object A in a system and another user tries to store an object B having content identical to object A, a match will be detected as the content hashes of objects A and B will be the same. Thus, a new object B is not created when a write-once access property is enforced.

[0031] If a write-many property is enabled *314*, high-level OID table *200* is consulted to determine whether a particular LUN/LBA *202* is associated with an OID *204*. If LUN/LBA has a null OID *316*, write operation is to LUN/LBA *332* is allowed. Subsequent to write operation to LUN/LBA *332*, OID is written to high-level table *334*. If LUN/LBA has an OID

310, then application program data *302* is allowed a re-write operation to LUN/LBA *320* and a previous OID is over-written in a high-level OID table *322*.

[0032] If a write-many with versioning property is enabled *324*, high-level OID table *200* is consulted to determine whether a particular LUN/LBA *202* is associated with an OID *204*. If LUN/LBA has a null OID *326*, write operation is to LUN/LBA *332* is allowed. Subsequent to write operation to LUN/LBA *332*, OID is written to high-level table *334*. If LUN/LBA has an OID *328*, then application program data *302* is allowed a write operation to LUN/LBA and an OID associated with application program data *302* is added to the head of a list in an OID table *330*. A previous OID moves toward the tail of the list. If a newly generated OID already exists within an OID list, then a counter for that OID is incremented to prevent its accidental deletion. When content data associated with a particular OID is deleted by an application, if an associated counter value is not zero, then that particular content data is not deleted from its physical storage location because another application could be using that content data.

[0033] A user may also control the granularity of the access type chosen granularity occurs at levels including an entire

system, an aggregation of LUN/LBAs, and a single LUN/LBA. Applications determine what elements to aggregate and the level of aggregation.

[0034] Referring now to Fig. 4, a second tier logic block is shown. Second tier OID processing *400* receives content reference data from a first tier interface *402* or directly from an application *404*. Thus, an application can directly opt to use OID interface of second tier *400* instead of accessing data via LUN/LBA interface of first tier *300*. In second tier OID processing *400*, content reference data to be written is hashed and an OID is generated *406*. Next, a high-level OID table is updated *408* with newly generated hash value for an associated LUN/LBA *202*. Low-level OID table is further updated *410* with newly generated hash value. Second tier OID processing *400* helps to eliminate duplicates - if two separate LUN/LBA combinations have the same data content, then content associated with both combinations will hash to the same OID. LUN/LBA counter is incremented *412*.

[0035] Referring now to Fig. 5, a third tier logic block is shown. Third tier OID processing *500* receives data from second tier interface *502*.Low-level OID table is accessed *504* to write to an LUN/LBA *506* data received from second tier

interface *502*. If data from a particular LUN/LBA combination is to be retrieved *508*, reference data it is re-hashed and the hash value generated is compared against an OID value *510* generated during the original writing of data in second tier logic block *400*. In this tier, data is simply written or retrieved from a lower level LUN/LBA of a storage subsystem. Thus, third tier OID processing *500* helps to verify the accuracy of lower level LBA content that has been retrieved *508* from a storage subsystem.

[0036] Shown in Fig. 6 is a data flow diagram for writing content data. Content data and an LUN/LBA address combination are provided as data from application program *602* to first LUN/LBA processing tier *604*. First LUN/LBA processing tier *604* creates, if necessary, an entry in a high-level OID table for an LUN/LBA address combination provided as data from application program *602*. Data passed between first LUN/LBA processing tier *604* and second OID processing tier *606* comprises content data and an LUN/LBA element location address from a high-level OID table. Second OID processing tier *606* hashes content data received from first LUN/LBA processing tier *604* and generates an OID. Second OID processing tier *606* fills in generated OID value in a high-level OID table at an LUN/LBA el-

ement location address also received from first LUN/LBA processing tier *604*. Content data, an OID, and a location of the OID in a low-level OID table are passed to a third storage subsystem LUN/LBA processing tier *608*. After writing content data to a physical LUN/LBA combination, third storage subsystem LUN/LBA processing tier *608* fills in a physical LUN/LBA combination in a low-level OID table in an entry corresponding to OID of written content data.

[0037] Shown in Fig. 7 is a data flow diagram for retrieving content data. Application program data *702* provides a LUN/LBA address combination from which to retrieve content data. At first LUN/LBA processing tier *704*, a high-level OID table is consulted to determine an OID corresponding to an LUN/LBA combination received from application program data *702*. First LUN/LBA processing tier *704* passes an OID to second OID processing tier *706*. At second OID processing tier *706*, a low-level OID table is consulted to determine a physical LUN/LBA combination corresponding to an OID received from first LUN/LBA processing tier *704*. A physical LUN/LBA combination is passed to third storage subsystem LUN/LBA processing tier *708* to retrieve content data. Content data is passed

from third storage subsystem LUN/LBA processing tier *708* to second OID processing tier *706* to first LUN/LBA processing tier *704* and finally, to application program *702* requesting the retrieval.

[0038] The present invention does not preclude another application from simultaneously using the OID type interface to get the benefits of a larger address space of the OID interface. That is, one application may interface at a first LUN/LBA processing tier, whereas another application may interface at a second OID processing tier, thereby bypassing a first LUN/LBA processing tier.

CONCLUSION

[0039] A system and method has been shown in the above embodiments for the effective implementation of a flexible LUN/LBA interface for content addressable reference storage. While various preferred embodiments have been shown and described, it will be understood that there is no intent to limit the invention by such disclosure, but rather, it is intended to cover all modifications and alternate constructions falling within the spirit and scope of the invention, as defined in the appended claims. For example, the present invention should not be limited by application software/program or specific computing hard-

ware or peripheral device.

[0040] All programming, GUIs, display panels and dialog box templates, and data related thereto are stored in computer memory, static or dynamic, and may be retrieved by the user in any of: conventional computer storage, display (i.e. CRT) and/or hardcopy (i.e. printed) formats. The programming of the present invention may be implemented by one of skill in the art of object-oriented programming.